**STREAMS**

**Definición**

**Qué es un Stream**

Un Stream es un medio utilizado para leer datos de una fuente, y para escribir datos en un destino. Tanto la fuente como el destino pueden ser archivos, sockets, memoria, cadenas de caracteres, y también procesos.

Los Streams se caracterizan por ser unidireccionales, es decir que un Stream se utilizara solo para leer, o solo para escribir, pero no ambas acciones al mismo tiempo.

Para utilizar una Stream, el programa a realizar deberá construir el Stream relacionándolo directamente con una fuente o con un destino, dependiendo si se necesita leer o escribir información.

La acción de leer información de una fuente es conocida también como input, y la acción de escribir información en un destino es conocida como output. Dentro de Java, todas las clases utilizadas tanto para el input como para el output están incluidas en el paquete java.io

**Algoritmo de Lectura**

Para obtener información, un programa deberá abrir un stream sobre una fuente y leer la información de forma secuencial.

Independientemente del tipo de información, el algoritmo de lectura es siempre el mismo:

Abrir un stream

Mientras haya más información

Leer información

Cerrar stream

**Algoritmo de Escritura**

Para escribir información, un programa deberá abrir un stream sobre un destino y escribir la información de forma secuencial.

Independientemente del tipo de información, el algoritmo de escritura es siempre el mismo:

Abrir un stream

Mientras haya más información

Escribir información

Cerrar stream

**Tipos de Streams**

**Organización**

La tecnología Java contiene distintos tipos de Streams, lo cuales estan organizados en dos grandes grupos:

* Streams orientados a Caracter (Character Streams)
* Streams orientados a Byte (Byte Streams)

**Streams orientados a Caracter**

Los Streams orientados a Caracter operan con caracteres como unidad de trabajo. Los caracteres a leer están formados por 2 bytes (es decir 16 bits por carácter).

Son utilizados para leer y escribir información que está almacenada en forma de texto, como por ejemplo archivos de extension txt, ini, csv, etc.

La superclase utilizada para leer streams orientados a caracter es la clase Reader. A partir de esta clase – la cual es abstracta – heredan todas las clases concretas que se utilizan para leer información en forma textual.

Por otra parte, la superclase utilizada para escribir streams orientados a caracter es la clase Writer. A partir de esta clase – la cual es abstracta – heredan todas las clases concretas que se utilizan para escribir información en forma textual.

**Streams orientados a Byte**

Los Streams orientados a Byte operan con bytes como unidad de trabajo. Los bytes a leer se leen en forma unitaria (es decir 8 bits por byte).

Son utilizados para leer y escribir información que está almacenada en forma binaria, como por ejemplo archivos de extensión jpeg, mpeg, xls, etc.

La superclase utilizada para leer streams orientados a byte es la clase InputStream. A partir de esta clase – la cual es abstracta – heredan todas las clases concretas que se utilizan para leer información en forma binaria.

Por otra parte, la superclase utilizada para escribir streams orientados a byte es la clase OutputStream. A partir de esta clase – la cual es abstracta – heredan todas las clases concretas que se utilizan para escribir información en forma binaria.

**Qué es un File Stream**

**Definición**

Los File Streams son los streams utilizados para lectura y escritura de (particularmente) archivos, es una categoría que agrupa tanto a los streams orientados a caracter como a los streams orientados a byte.

En general se utilizan en conjunto con un objeto del tipo File, que es una representación abstracta de un archivo. La clase File modela tanto archivos como directorios.

**Lectura de un Archivo de Texto**

La clase FileReader es una clase concreta utilizada para generar streams orientados a carácter, y es la encargada de realizar la lectura de archivos en forma de texto.

A continuación, se presenta un ejemplo de lectura de un archivo llamado fuente.txt:

import java.io.\*;

public class Lector

{

public static void main(String[ ] args) throws IOException

{

// Define el archivo a utilizar

               File archivoEntrada = new File("fuente.txt");

               // Abre el stream necesario

FileReader in = new FileReader(archivoEntrada);

int unCaracter;

               // Lee el archivo

while ( (unCaracter = in.read()) != -1)

System.out.print((char)unCaracter);

// Cierra el stream

in.close();

}

}

**Escritura de un Archivo de Texto**

La clase FileWriter es una clase concreta utilizada para generar streams orientados a carácter, y es la encargada de realizar la escritura de archivos en forma de texto.

A continuación, se presenta un ejemplo de escritura de un archivo llamado destino.txt:

import java.io.\*;

public class Escritor

{

public static void main(String[ ] args) throws IOException

{

// Define el archivo a utilizar

File archivoSalida = new File("destino.txt");

// Abre el stream necesario

FileWriter out = new FileWriter(archivoSalida);

// Define la información a guardar en el archivo

String info = “Soy la informacion”;

// Escribe el archivo con la información

for(int i=0; i<info.length(); i++)

out.write( info.charAt(i) );

// Cierra los streams

out.close();

}

}

**Lectura y Escritura de Archivos Binarios**

La clase FileInputStream es una clase concreta utilizada para generar streams orientados a byte, y es la encargada de realizar la lectura de archivos en forma binaria.

La clase FileOutputStream es una clase concreta utilizada para generar streams orientados a byte, y es la encargada de realizar la escritura de archivos en forma binaria.

A continuación, se presenta un ejemplo de lectura de un archivo llamado fuente.gif y la escritura de dicho contenido en un archivo llamado destino.gif. La clase Copiador presentada a continuación lee la imagen, y genera un duplicado de la misma:

import java.io.\*;

public class Copiador

{

public static void main(String[] args) throws IOException

{

               // Define los archivos a utilizar

               File archivoEntrada = new File("fuente.gif");

               File archivoSalida = new File("destino.gif");

               // Abre los streams necesarios

FileInputStream in = new FileInputStream(archivoEntrada);

FileOutputStream out = new FileOutputStream(archivoSalida);

int unCaracter;

               // Copia el archivo fuente en el archivo destino

while ( (unCaracter = in.read()) != -1)

out.write(unCaracter);

// Cierra los streams

in.close();

out.close();

}

}

**Qué son los Buffers**

**Definición**

Los buffers son una alternativa a las clases básicas de entrada y salida. Son subclases de las clases básicas correspondientes a la clase Reader, Writer, InputStream y OutputStream.

Cumplen el mismo objetivo que dichas clases, pero son eficientes ya que tienen como objetivo guardar en un buffer los caracteres leídos / por escribir para lograr una mejora sustancial en la lectura / escritura.

Son utilizados como wrappers (envoltorios) para envolver las clases básicas.

**La clase BufferedReader**

La clase BufferedReader es una clase que hereda de la clase Reader y se utiliza para envolver a otras subclases del tipo Reader, tales como la clase FileReader.

Entre los métodos más utilizados se encuentra el método readLine(), que permite leer un conjunto de caracteres retornado en forma de String, en lugar de leer caracter a caracter.

La clase BufferedReader puede ser utilizada de la siguiente manera:

// Instancia un objeto del tipo BufferedReader llamado in

File archivo = new File(“fuente.txt”);

BufferedReader in = new BufferedReader(new FileReader(archivo) );

// Lee la primer linea del archivo fuente.txt a traves del stream

// denominado in

String lineaLeida = in.readLine();

// Cierra el stream y libera recursos

in.close();

**La clase BufferedWriter**

La clase BufferedWriter es una clase que hereda de la clase Writer y se utiliza para envolver a otras subclases del tipo Writer, tales como la clase FileWriter.

Entre los métodos más utilizados se encuentra el método write(String s, int offset, int length), que permite para escribir un conjunto de caracteres en lugar de leer caracter a caracter.

Adicionalmente, posee un método denominado newLine() que se utilizada para escribir en el stream la representación de una nueva línea, es decir la tecla “enter”, entendiendo que cada sistema operativo puede representarla con distintos caracteres.

La clase BufferedWriter puede ser utilizada de la siguiente manera:

// Instancia un objeto del tipo BufferedWriter llamado out

File archivo = new File(“destino.txt”);

BufferedWriter out = new BufferedWriter(new FileWriter(archivo) );

// Linea a escribir

String linea1 = "Hola, soy una línea a escribir en el archivo";

// Escribe la cadena de caracteres en el archivo

out.write(linea1, 0, linea1.length());

// Escribe un “enter” en el archivo

out.newLine();

// Cierra el stream y libera recursos

out.close();

**La clase BufferedInputStream**

El objetivo de la clase BufferedInputStream es el mismo que el de la clase BufferedReader, la diferencia radica en que el tratamiento de la información es a nivel bytes, y no a nivel caracteres.

**La clase BufferedOutputStream**

El objetivo de la clase BufferedOutputStream es el mismo que el de la clase BufferedWriter, la diferencia radica en que el tratamiento de la información es a nivel bytes, y no a nivel caracteres.